An algorithm for optimized generation of a finite element mesh for folded airbags

A Chawla
(Corresponding Author)
Professor
Department of Mechanical Engineering
Indian Institute of Technology, New Delhi
Tel No.: 011-26591058
Fax No. : 011-26582053
Email: achawla@mech.iitd.ernet.in

S Mukherjee
Professor
Department of Mechanical Engineering
Indian Institute of Technology, New Delhi

A Sharma
Department of Mechanical Engineering
Indian Institute of Technology, New Delhi

ABSTRACT

This work presents a novel algorithm for generating a geometric mesh of an airbag after a series of folds are defined on it, for use in crash simulations. The process of airbag folding is simulated as a series of geometric transformations to the airbag mesh, which is modeled as a stack of connected planar layers. Along with these transformations, optimization techniques are used to minimize change in the geometry and the area of the airbag post inflation. The results from this approach are compared with commercially available airbag folding software. It is observed that the algorithm is effective in limiting change in the area and geometry of the airbag in the folding process. The change in surface area is only 0.1% of the surface area of the airbag against the change of 11% in the commercial package. The current paper is a sequel to [3] which reports initial results of the folding process.

Keywords: airbag modeling, finite element simulation, mesh generation

1 INTRODUCTION

Airbags have been shown to be a vital safety device in vehicle crashes worldwide. When isatalled, the airbag is folded into a small volume and connected to an inflator, which infuses gas generated from an explosive chemical reaction into the airbag in the event of a crash. Thus, to simulate the behavior of the airbag, a model of the airbag in the initial folded position is required.. The duration from the initial impact during a crash to the full inflation of an airbag is about 40 milliseconds and during this time, the airbag goes from being in a folded state to a fully inflated state, with a high internal pressure. After achieving this state, the airbag begins to deflate, thus reducing the internal pressure and providing a cushion for the body impacting it. Ideally the occupant contact should contact a fully inflated airbag. In the field, contact may take place before the airbag is fully inflated or the occupant may initially impact the airbag at its periphery instead of the center of the airbag, reducing the protection level. In these non-desirable contact positions, the airbag unfolding process would determine the extent of safety provided to the passenger, as the state of the airbag at intermediate states would be different for varying folds used to pack them. Thus, while studying crashes using simulations, the behavior of the airbag when unfolding also needs to be modeled. Some of the available software for vehicle crash modeling, such as PAM-CRASH^{TM} [7], have modules for folding airbags. The airbag-modeling module in the PAM-CRASH^{TM} suite of software is SAFE EDITOR^{TM} [8], which allows a user to define airbag geometry and perform a series of folding operations on that geometry. After the user has defined the required folds on the airbag geometry, a mesh of the airbag can be generated in SAFE EDITOR^{TM}. The approach

taken by SAFE EDITOR^{TM} is to model the airbag as a geometric surface, which is transformed to achieve a folded state of the surface. This transformation is dependant on a number of parameters that are enumerated in detail in later sections. The folded (transformed) geometry of the airbag is then meshed and can be input for solving in an FE solver. The surface transformation mentioned above is computationally expensive to perform and hence SAFE EDITOR^{TM} restricts the airbag geometry to simple surfaces. One could presume that, in order to keep the geometry simple during folding, and to prevent the transformation from becoming computationally infeasible, it compromises on the crispness of the fold definition. These could be limiting when generating realistic models of the airbag in use today which demand to be modeled using a larger set of surfaces with crisper fold definition.

1.1 Applications for cloth models in computer graphics and animation industry

Cloth surfaces have been modeled using the spring-mass models by Zhang [6], with self collision modeling being incorporated by Ng [10], [11], and recently wrinkle modeling by Bridson [1]. Ng [10] offers an excellent survey of computer graphics techniques used to model cloth. The emphasis in recent years have been on physically based models of cloth, incorporating cloth dynamics and modeling the cloth as a very fine mesh, whose behavior is governed by self collisions and interaction of cloth with other materials. One such sophisticated technique is presented by Bridson [1] which has been successfully applied to computer animation. Thus, the emphasis in cloth modeling has been on obtaining a realistic looking model of the cloth rather than using it in Finite Element (FE) simulations. The applications of the model are mostly found in the animation industry. Hence, the models are either too refined to be of use in FE simulations (the computation time increases exponentially with refinement in a FE mesh), or, are unable to handle very tight squeezing of the cloth, which is required in airbag folding. As an illustrative example, in a driver side airbag, used as a case study for this work, the airbag cloth is a 600mm diameter membrane and is packed in a casing which approximates a cuboid of size 130 × 110 × 30 mm. This results in a very tight squeezing of the airbag surface against itself.

In an explicit FE simulation package like PAM-CRASH, the computational time is linearly dependent on the number of degrees of freedom in the model, and on a worst case timestep determined by the size and shape of the elements. The more refined the mesh, the smaller the minimum timestep and higher the computation time. This work addresses the need for a model of an folded airbag, which can model the tight folds of the airbag, and still be computationally feasible in FE simulations. Airbag folding for FE simulation has some specific issues which need to be highlighted. Naïve folding algorithms tend to generate a fine mesh near the fold to enable exact representation of the fold line. In case of the airbag, it is not the folded membrane but the final unfolded membrane that is of interest. Secondly the unfolded volume needs to expand into the same shape and volume as the actual device. This is a special problem with quadrilateral shell elements as preserving surface area under out of plane warping, and preserving element edge lengths under warping post folding is challenging. The third issue of significance is that penetrations between inner layers have to be avoided. Any internal penetration, though not visible as an artifact on the surface, has catastrophic consequences in FE simulations of airbag unfolding. The final issue is that the element sizes have to be above a minimum threshold to ensure stable simulations with explicit solvers being completed in acceptable time.

The usual approach to airbag folding is a geometric one, consisting of equations describing the airbag surface, which is meshed after the completion of the folds. We demonstrate a novel airbag folding approach, which folds a given initial 3D mesh of an airbag multiple times and refines the mesh as per the requirements of the fold. Compared with the PAM-SAFE package for airbag folding, it is observed that the change in area and geometry of the airbag is substantially reduced by this technique.

Initial results from this algorithm in [3] have now been fully developed and tested. The folding process itself is modeled as a geometric transformation combined with an optimization step to locate critical points in the fold. The optimization being to minimize the change in geometry and surface area of the airbag. After the final fold, a FE mesh of the airbag is obtained by assigning physical properties to the airbag material. This FE model can be imported like any other mesh and inflated using a FE airbag simulator package like PAM-CRASH.

2 PROBLEM DEFINITION

Our objective is to generate a 'well-behaved' geometric mesh of a given airbag in a folded position, given the initial geometry of the airbag and the sequence of folds that are defined on it. A well-behaved mesh of an airbag being a mesh that is amenable to computation using FE simulations. For example, a mesh with very small elements or elements with a very high (or very low) aspect ratio is not well behaved as the time for computation would increase. The initial geometry of the airbag is constrained to be in the form of a set of parallel planar layers, which are connected using an initially smaller set of elements arranged in a planar configuration (the connecting planes being inclined to the parallel layers and are referred to as inclined layers in this text). This initial simplification is imposed as non-inflated airbags can be 'patted down' flat to this configuration which is the atrt of the airbag folding process.

The initial geometry is then meshed using shell elements, which are either quadrilateral or triangular. These are also the two kinds of elements available to model planes in FE packages and suffice to model generic surfaces. Hence the package has been limited to handle only these kinds of elements. This initial mesh is input to our package, and the user is asked to input the values for the parameters, which are used to define the fold. Fold definitions are not standardized and different commercial packages use their own parameters to define folds. We evolved a set of parameters for defining a fold which take into account the positioning and tightness of the fold, as well as specifying which segment of the cloth is to be repositioned during a fold. The algorithm then proceeds to refine the mesh to enable the fold to be carried out on the mesh and transforms the elements to their new positions, which are governed by the parameter controlling the tightness of the fold. Thus, the configuration of the mesh after the fold is determined and output in form of a mesh. This process of defining and executing folds can be continued till the requisite configuration is achieved. The initially output mesh may have some elements with a non-desirable aspect ratio and size, and for correcting these, there is a 'coarsening' module in the package, which inputs the mesh along with the required refinement level and outputs a mesh better suited for FE simulations .

3 EXISTING TOOLS

PAM-CRASH^{TM} software [7] has a module for folding airbags, called SAFE EDITOR^{TM} [8], which inputs the initial geometry of the airbag and generates the geometry resulting after a sequence of folds. One can also define gas properties and other parameters required for airbag inflation in SAFE EDITOR^{TM}, but for this discussion only the airbag meshing options in the software is relevant. This module is a geometry-based folder, which takes the initial airbag geometry as input and allows users to define folds on the airbag, giving the mesh of the final geometry as the output. SAFE EDITOR^{TM} takes an IGES geometry file as an input for the initial geometry. In this case study, the IGES file given as input to the airbag folder, contains the geometry of a circle. As the SAFE EDITOR^{TM} takes only a planar geometry as the initial input, it is constrained to having the lower layer of the airbag identical to the upper one connected by common edges. After giving the initial geometry, one can proceed to define folds on the airbag. The software supports two kinds of folds: the roll fold (shown in Figure 1) and the tuck fold (shown in Figure 2)..

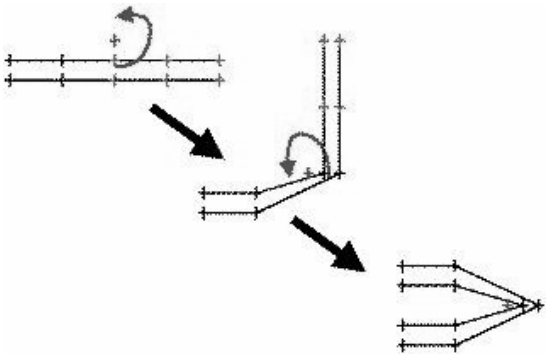

Figure 1 A roll fold

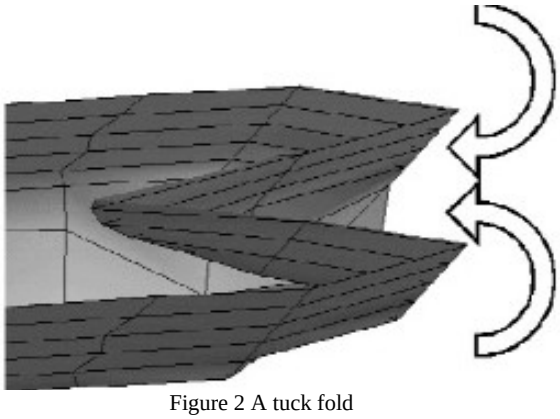

Figure 2 A tuck fold

The parameters to be input for defining a roll fold in SAFE EDITOR are location of the folding line, direction of the fold, a fixed point, thickness of the fold and the transient distance for the fold. Thus, one defines a folding line only on one surface of the airbag, which is propagated as the folding location for the rest of the airbag as well. The direction of the folding line can be reversed to change the direction of the fold from clockwise to anti-clockwise. The transient distance of an airbag fold corresponds to the width of airbag fabric on both sides of a folding line that will participate to the fold. The fixed point is a point that identifies the airbag patch that remains fixed during a folding sequence. The thickness of the fold or the outside thickness is defined as the desired distance between the innermost layers after the fold. An illustration of these parameters is provided in Figure 3. The generation of mesh is a separate process from the folding in SAFE EDITOR, and is not linked to the folding sequence. Thus the folding sequence can be changed quickly, without having to mesh the geometry at each stage, as only the final state can be meshed to obtain a "pc" file, which can then be input for FE simulation in PAM-CRASH.

There are restrictions for defining folds in SAFE EDITOR^{TM}. The successive folding lines have to be either parallel or orthogonal to each other for defining a fold. Both the folding lines and transient lines have to be defined by points lying outside the airbag and are always applied to the full stack of layers. Also, a transient patch cannot be split by another parallel folding or transient line, even if only the transient line divides a given layer (since folds are applied to the full stack).

4 DESCRIPTION OF PACKAGE

As stated earlier, the approach proposed here views the folds as a series of geometric transformations applied on the airbag mesh, coupled with an optimization approach to locate critical points in the mesh. This algorithm mimics a real-life folding process, wherein the cloth is folded physically by locating a folding line and then rotating a part of the cloth such that it lies on the top of the other. When the cloth is physically folded, wrinkles are generated on the cloth surface. Wrinkles have been modeled through computationally expensive processes in the past because they constitute an important look-and-feel component in textiles. For FE simulation of the inflation process, they could be ignored but for the loss in the area of the cloth surface, resulting in distortion of the geometry of the airbag post inflation. In the proposed folding procedure, the area preserving effect of wrinkles has been incorporated, without sacrificing computation time.

The initial airbag surface is defined as a set of connected planar layers. Thus, prior to defining a fold on the airbag surface, it has to be collapsed and flattened into a set of connected planar layers. There are alternate methods to achieve this flattening like physical measurements of the actual airbag, and simulated flattening of the airbag using flat surfaces or using an approximation mesh to the actual shape. In the present case study, an approximation mesh of the shape of a disc of 600mm diameter is being used, which is very close to the actual shape of the airbag. Thus, the flattening of the airbag could result in several parallel layers (in our case study, the two planar connected layers), which have to be transformed simultaneously to implement folds on the deflated airbag, while taking care to avoid mesh self-penetrations.

Each fold is defined by four input parameters:

1. The choice of a folding plane.
2. The desired width of the fold (which is the desired distance between the innermost layers after the fold).
3. The folding part (whether the mesh lying on the left or the right side of the folding plane is to be folded).
4. The folding direction (whether the mesh is to be folded clockwise or anti-clockwise).

Thus, at each step, an airbag mesh is given as input, on which the user (using the above parameters) defines a fold, and the algorithm generates an output mesh of the airbag in the folded position. In the process of folding, if the mesh needs to be refined, the algorithm takes this into account.

After each fold, the output is a mesh of the airbag in the desired folded position. Usually, after the fold, the mesh has a higher level of refinement (smaller elements) than the initial model. As a refined mesh becomes computationally expensive for FE simulations, there is a need to coarsen the mesh at the end of the folding process, while keeping the geometry intact, to make the model amenable to simulation using explicit FE packages. An algorithm for coarsening the mesh is also incorporated in the software.

Each fold of the airbag generates one or more planar layers, which are added to the already existing set of parallel layers, and connected to the rest of the mesh by inclined layers. The mesh can be folded again and again, as long as the layers are wide enough to support folding, though the time taken for the computation would increase with the number of planar layers. As the folding algorithm is modeled on the real world process of folding, the outer layers in a fold occupy a larger circumference compared to the inner layers in a fold. Thus, each layer that is folded, forks into two layers, both of which are smaller than the original layer, and two inclined layers.

These rotations are shown in Figure 4 with the help of an example of a disc being folded. The subfigures represent the intermediate stages of the folding process.

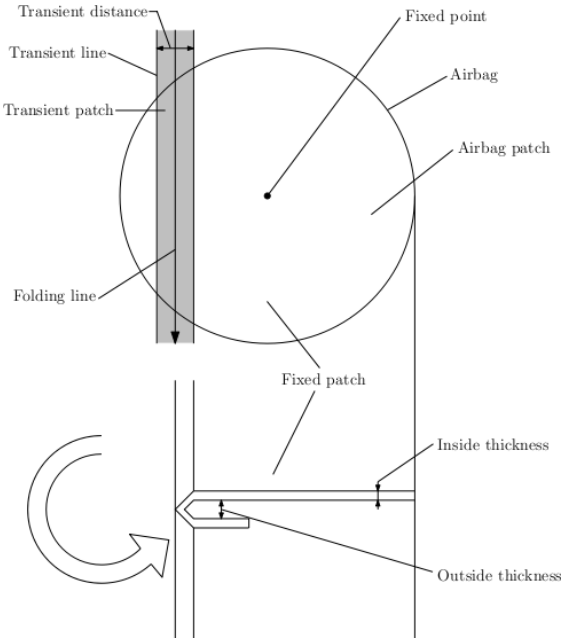

Figure 3 Illustrations of the parameters for defining a roll fold

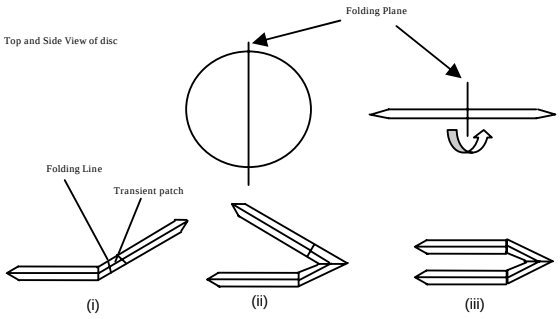

Figure 4 Illustrations of Folding Transformations, with intermediate steps in folding

The area of each individual layer keeps on decreasing with each fold, though the algorithm approximately preserves the sum of all the areas. And after a large number of folds, the layers may become too small to be able to accommodate a transient patch, signaling that no more folds can be carried on it. This process is intuitively similar to the real world folding process, where a cloth can be folded only a certain number of times till the layers become too small to be folded again. The numbers of folds for the examples we have considered are quite large and easily satisfy our requirements for airbag folding.

Maintaining the sequence of folds is also critical, as different sequences result in different unfolding patterns, thereby affecting the deployment of the airbag during inflation.

This algorithm is capable of handling folds in arbitrary directions, provided fold lines do not intersect some special sections of the mesh called transient patches. Transient patches are created in zones where the folding plane intersects the inclined layers and the creation of these patches is necessary to avoid change in the surface area of the airbag. This is not a serious functional limitation as transient patches created by the algorithm are very small compared to the airbag area. For practical purposes, there is no restriction on the direction of folding planes, as long as they do not intersect the transient patches, giving a lot of flexibility in trying out various folding sequences.

5 FOLDING ALGORITHM

The folding process is visualized as a series of geometric transformations applied on the input mesh in order to achieve a desired state. The mesh must have all nodes lying on a finite number of planes, which are henceforth referred to as layers. These layers can exist only in two possible configurations, one in which there are two layers nominally parallel to each other and containing the majority of the nodes. These layers are termed parallel layers. The second set of layers are called inclined layers, which connect the parallel layers such that two parallel layers are joined using pairs of inclined layers. There is at least one pair of inclined layers joining every parallel layer with at least one other parallel layer. The concept of inclined layers is illustrated in Figure 5, in which several pairs of inclined layers connect a set of four parallel layers (out of which for two layers only the edges are visible in the figure).

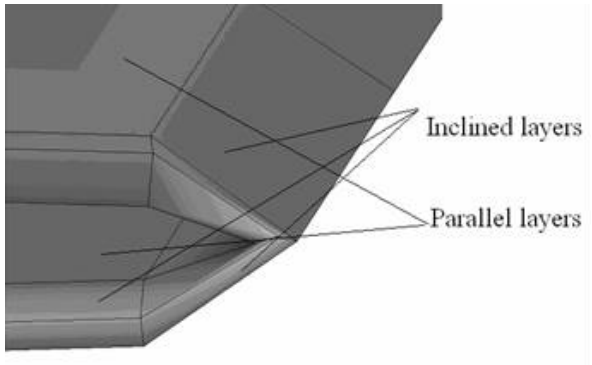

Figure 5 The parallel and inclined Layers

Thus all the nodes in the mesh can be classified as being either on one of the parallel layers or on one of the inclined layers, with the common nodes lying on the intersection of the parallel and inclined layers being listed with the planar layers. The inclined layers are further classified as the “Left” and the “Right” inclined layers, with the words indicating whether the inclined layer is connected to the upper parallel layer or the lower parallel layer. The nomenclature has syntactic relevance and should not be concocted to as having the usual physical meaning. Once the inclined layer is tagged; a consistent convention is followed throughout the folding process. The nodes lying on the intersection of left and right inclined layers are listed in the left inclined layers, and can be easily found if required for the right inclined layer, as each combination of left and right inclined layer has the same list index. The planes for all the layers (parallel and inclined) are also stored in the explicit form [A B C D], where $A.x + B.y + C.z + D = 0$ is the equation of the three dimensional plane.

Once the user inputs the folding plane, the folding width, transient distance, the folding direction and the folding part, the parallel layers are sorted in order of their stacking, which is determined by the folding direction (the ordering reverses for contra-parallel fold lines) and distance between the layer planes, all of which are parallel. Then, three planes split each connected parallel layer pair, starting from the topmost layer in the ordering. These three planes are different for each layer pair and are determined from the location of the folding plane, the value of the folding width and the transient distance. An illustration of the three planes is given in Figure 6. After the planes split the mesh, the parallel layers can be rotated to

achieve the folded position of the mesh. The inclined layers are placed in position using routines that use a combination of geometric transformations and optimization techniques to achieve folding. The outline of the algorithm is described below, and the details of each step of the algorithm are provided in subsections following the outline:

Step 1. INPUT STAGE:

Input the following:

- Folding plane,
- Folding Width (w),
- Folding direction (FD - Clockwise (1) or Anti-Clockwise(-1))
- Folding part (FP - Left (1) or Right (2))

Step 2. PRE PROCESSING

2.1. Sort all parallel layers in the order of their stacking: $\{PL_1, PL_2, ..., PL_n\}$.

2.2. Classify parallel layers into "foldable" or "static" based on FP value.

2.3. Further classify "foldable" parallel layers into "split" or "rotated" based on its location relative to the folding plane.

2.4. Classify inclined layer pairs $\{IL_1, IL_2, ..., IL_m\}$ into four categories based on their position: "Left Untouched", "Split", "Mapped" or "Rotated".

Step 3. SPLIT TOP MOST PARALLEL LAYER

3.1. Find (PL_k) such that $k = \min\{i : (PL_i \in "foldable") \wedge (PL_i \in "split")\}$. [PL_k is the topmost parallel layer that is to be split]

3.2. Find (PL_t), where $t = \min\{i : PL_i \in "foldable"\}$, do { [PL_t is the topmost layer]

- Find the distance (h) between PL_k and PL_t.
- Calculate Transient Distance (TD) for this layer $TD = 2h$.
- Construct a "Global Left Offset Plane" ($GLOP$), which is given by $ax + by + cz + d' = 0$ if the folding plane is given by $ax + by + cz + d = 0$, where $d' = d - (-1)^{FP} * TD$.
- Split entire mesh with $GLOP$ (Section 5.3).

Step 4. PRE_PROCESSING:

For each inclined layer IL_i, where $IL_i \in "Split"$, {Call Fold_inclined_layer_pre-processing (Section 5.1}

Step 5. PARALLEL LAYER FOLDING:

For each parallel layer PL_i, where $PL_i \in "foldable"$, do the following

5.1. Find the distance (h) between PL_i and PL_t.

5.2. Calculate Transient Distance (TD) for this layer, given by $TD = 2h$.

5.3. Calculate the inclination angles: $\theta_1 = \sin^{-1}\left(\frac{h}{TD}\right)$ and $\theta_2 = \pi - 2\theta_1$.

5.4. Construct two planes: "Middle Plane" (MP) and "Right Offset Plane" (ROP) parallel to the "Global Left Offset Plane" as follows:

- MP is given by $ax + by + cz + d_m = 0$, where $d_m = d + (-1)^{FP} * 2 * TD$, FP is the folding part value and is either 1 or 2.
- ROP is given by $ax + by + cz + d_r = 0$, where $d_r = d + (-1)^{FP} * 4 * TD$, FP is the

folding part value and is either 1 or 2.

5.5. Split PL_i with the three planes: “Left Offset Plane”, “Middle Plane” and “Right Offset Plane” (Section 5.3).

5.6. Make three rotation axes, each one of which is the intersection line of two planes:

- Axis-1 ($GLOP$ & LP), Axis-2 (MOP & LP) and Axis-3 (ROP & LP).

5.7. Divide the nodes lying on PL_i into three sets:

- $NL_1 = \{node : [(node \in PL_i) \wedge ((node \in RightGLOP) \cap (node \in LeftMOP))]\}$,

  where, $Right / LeftGLOP$ indicates a set of nodes lying to the Right / Left of the Plane $GLOP$.
- $NL_2 = \{node : [(node \in PL_i) \wedge ((node \in RightMOP) \cap (node \in LeftROP))]\}$
- $NL_3 = \{node : [(node \in PL_i) \wedge (node \in RightROP)]\}$.

5.8. Rotate $NL_1 \cup NL_2 \cup NL_3$ by angle $FD * \theta_1$ about Axis-1.

5.9. Rotate $NL_2 \cup NL_3$ by angle $FD * \theta_2$ about Axis-2.

5.10. Rotate NL_3 by angle $FD * \theta_1$ about Axis-3.

5.11. Create a new parallel layer PL_{n+r} which consists of nodes NL_3.

5.12. Create a new inclined layer pair IL_{m+r} which consists of nodes NL_2.

Step 6. INCLINED LAYER FOLDING

For each inclined layer IL_i, where $IL_i \in "Split"$, {Call Fold_inclined layer subroutine (Section 5.2}

Step 7. POST PROCESSING

Recalculate the planes of all the layers (both parallel and inclined), which were transformed and post-process the data structures.

7.1. Run merger routine (Section 5.4)

7.2. Run penetration check (Section 5.5)

7.3. Run convexity check (Section 5.6)

5.1 The Fold Inclined Layer Pre-Processing Routine

This routine processes the inclined layers before they are folded. The algorithm is detailed as follows:

- Determine the location of the three planes that will split this inclined layer (as in the case of parallel layers).
- Split the inclined layer using the three planes and locate the nodes A, B, C, D, E, F, G, H, I, J, K and L. (Figure 7)

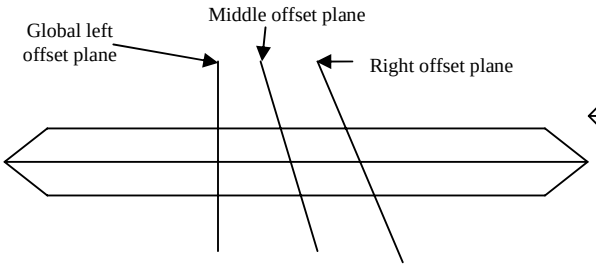

Figure 6 The three splitting planes

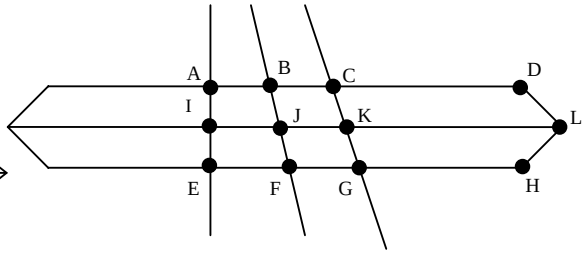

Figure 7 Nodes to be determined while folding inclined layers

The positions of the nodes A, B, C, D, E, F, G and H after folding lie on parallel layers after roation and can be calculated.

The node I would be stationary in the folding process as it lies on the folding axis and the mesh lying to the right of I is to be folded. Thus the new positions of the nodes J, K and L have to be determined in order to locate the folded position of the inclined layer in the mesh. We can visualize this as the original pair of inclined layer being split in four pairs (just as the parallel layer was split into 2 parallel layers and 1 pair of inclined layers), two of which are stationary as they lie to the left of the folding plane. The, six new inclined layers (i.e. three pairs of inclined layers) which are to be folded are each bounded by four nodes: ABJI, BCKJ, CDLK, KLHG, JKGF and IJFE. These six quadrilaterals each determine a new inclined layer, as the positions of the four bounding nodes of an inclined layer (the vertices of the quadrilateral) are sufficient to define the inclined layer. Thus, if the new positions of the nodes J, K and L, can be calculated then the new location of each of the six inclined layers can be determined, including the coordinates of the nodes lying within each of the six quadrilaterals.

5.2 The Inclined Layer Folding Routine

This routine folds the inclined layers. The algorithm is detailed as follows:

- The input to this routine is the set of data structures, which are generated from the inclined layers pre-processing routine (Section 5.1).
- The locations of the nodes J, K and L (Figure 7) are calculated using an optimization approach which seeks to minimize the total change in surface area of the mesh in the folding process, while enforcing that the new inclined layers BCKJ and FGKJ are planar.
- A Sequential Quadratic Programming (SQP) optimization algorithm is used to determine the locations of the nodes J, K and L, while keeping the location of the remaining nodes fixed. The nodes contained on the surface of the eight afore-mentioned quadrilaterals are then mapped to the new location of the quadrilaterals using barycentric coordinates. Figure 8 shows the location of one such node after folding. The SQP algorithm provided by MATLAB^{TM} is used for solving this optimization problem, and as the number of variables being solved for is very small (location of a few nodes each time), the optimization routine converges in less than 30 iterations. Again, since the size of the problem is quite small, the entire procedure takes only a few seconds to converge on a standard PC. Note that we are not concerned with finding the global optimum here and in fact want to find a solution that does not distort the original geometry too much. Hence our interest is in finding a locally optimal solution, and the “goodness” of the solution is finally measured by the change in the geometry of the airbag after inflation.
- This completes the folding process, as the parallel layers have already been folded by the time this routine is invoked.

5.3 Splitting the Mesh

The three planes, calculated for folding each parallel layer, split the mesh by determining which elements are to be split by each plane, using a linear search of all the elements. The splitting is a process of continually refining the mesh in order to create nodes and elements as and when required. Henceforth whenever a reference is mentioned to “creating a node”, it implies adding a new entry in the “NodeMatrix” data structure. Similarly, a reference to “creating an element”, it implies adding a new entry in the “ElementMatrix” data structure. As there are only two types of elements used in the mesh, viz. triangular and quadrilateral, we can exhaustively enumerate the cases under which each element would be split. These cases are listed below:

1. Triangular Elements: There are two possible scenarios when a plane splits a triangular element. These are as following:
    i. Middle Split: In this case, the splitting plane passes through one of the nodes and splits the opposing edge, where a new node is created, thus splitting the triangular element ABC into two triangles – ADC and DBC (a new triangular element is created for this). The middle split is illustrated in Figure 9.

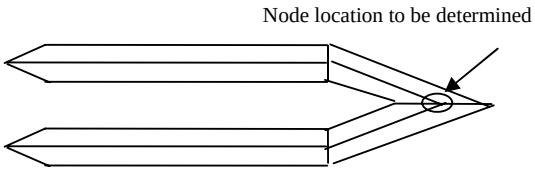

Figure 8 The node whose location is determined by minimization

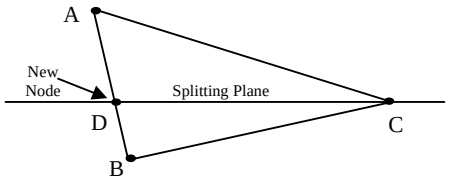

Figure 9 Top view of a triangular element split in the middle by a plane

ii. Side Split: In this case, the splitting plane intersects two edges of the triangle, creating two new nodes at the intersections, thus splitting the triangular element ABC into two elements: A triangle DBE and a quadrilateral ADEC, creating a new element in the process. The side split is depicted in Figure 10.

2. Quadrilateral Elements: There are four possible scenarios when a plane splits a quadrilateral element. These are as following:

i. Middle Split: In this case, the splitting plane intersects two opposing edges of the quadrilateral, creating two new nodes at the intersection of these edges with the plane, thus splitting the quadrilateral element ABCD into two quadrilaterals – AEFD and EBCF, creating a new quadrilateral element in the process. The middle split is depicted in Figure 11.

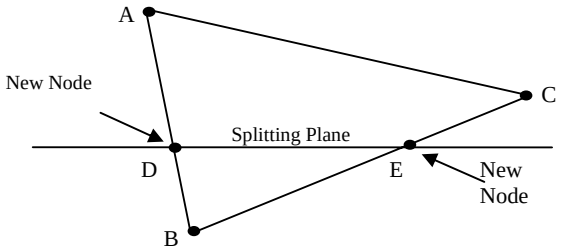

Figure 10 Top view of a triangular element split in the side by a plane

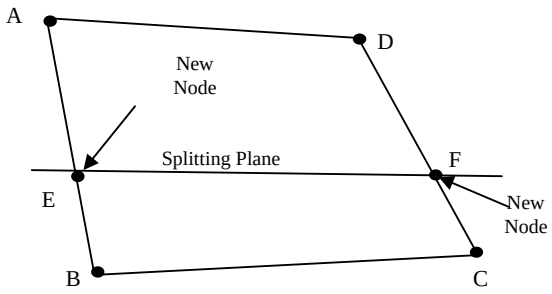

Figure 11 Top view of a quadrilateral element split in the middle by a plane

ii. Diagonal Split: In this case, the splitting plane passes through two opposing nodes of the quadrilateral, thereby splitting the quadrilateral element ABCD into two triangles – ABC and ACD, creating a new triangular element in the process. The diagonal split is depicted in Figure 12.

iii. Side Split: In this case, the splitting plane passes through a node of the quadrilateral and its opposing side, thereby splitting the quadrilateral element ABCD into two new elements: A triangle AED and a quadrilateral ABCE. The side split is depicted in Figure 13.

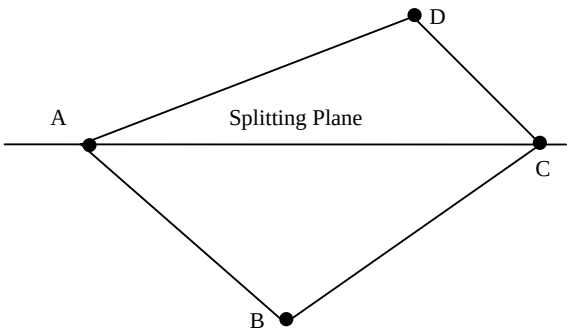

Figure 12 Top view of a quadrilateral element split diagonally by a plane

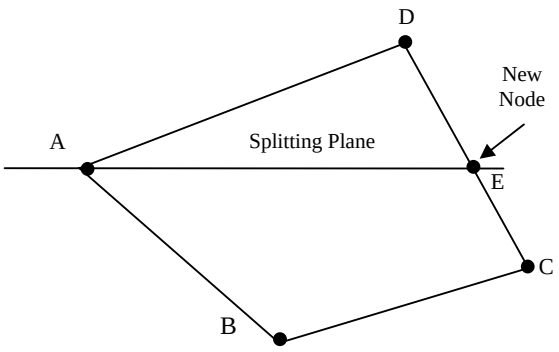

Figure 13 Top view of a quadrilateral element split in the side by a plane

iv. Both Side Split: In this case, the splitting plane intersects two adjacent edges of the quadrilateral, creating two new nodes at the intersection of these edges with the plane, thus splitting the quadrilateral ABCD into three new elements: Two triangles: FED and FCE, and a quadrilateral ABCF. The side split is depicted in Figure 14.

5.4 Merge Routine

A merge routine has been incorporated in the algorithm in order to coarsen the mesh according to the requirement of the user. The user specifies a tolerance of the edge length or the element area, so that any edge, which is shorter than the tolerance would be collapsed to a point or any element, whose area is less than the specified tolerance would be deleted. Thus, collapsing small elements and merging them into the neighbors achieve mesh coarsening. The neighboring elements to a small element that is merged are examined for the smallest area and merged with the collapsed element. The surface area of the mesh is not changed and coarsening is achieved. The meshing algorithm proceeds as follows:

- The input to this routine is the mesh itself, along with the threshold time-step (TS_0) for the mesh. The algorithm will coarsen the mesh until the desired time-step is matched or exceeded.
- For each element in the mesh, do the following:
  - Calculate the element time-step[7] :
  - Check whether the time-step is lower than the threshold time-step.
  - If ($TS < TS_0$) {Merge the element with one of its neighboring elements. Here, neighboring elements are those elements that share an edge with the element.}

5.5 Penetration Removal

The folding algorithm presented is a node geometry based algorithm, which does not consider interactions of specific nodes with elements that do not contain the node. Hence, the folding process may sometimes result in a physically infeasible configuration, such as the one shown in Figure 15 where one layer penetrates another. It is not physically possible for airbag surfaces to penetrate each other in this manner and hence this condition is to be avoided. Penetrations pose a horde of problems to any Finite Element solver and hence must be removed from an input model. An algorithm to detect and remove penetrations has been incorporated in the software to remove such penetrations from a folded airbag.

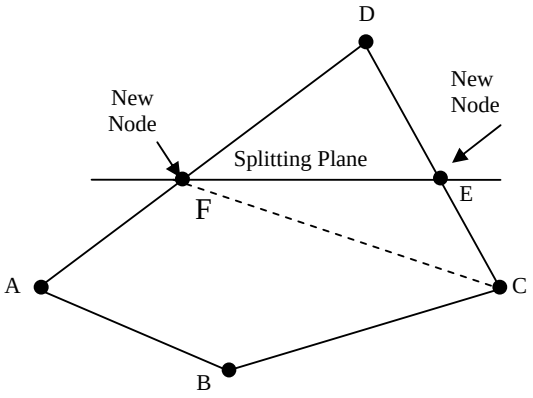

Figure 14 Top view of a quadrilateral element split in both sides by a plane

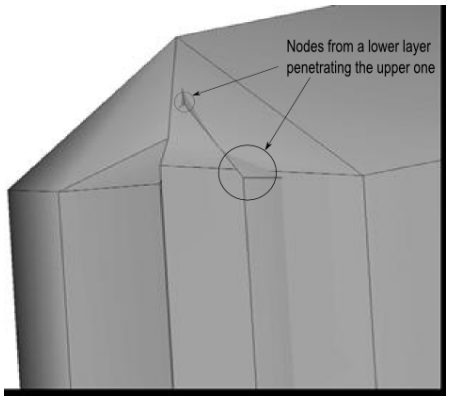

Figure 15 A penetration example

Only nodes lying on the intersection (boundary) of two inclined layers are candidates for penetration check.

This assumption is justified in the case of the present algorithm, as the nodes lying on parallel layers cannot penetrate each other, as the algorithm enforces a specified distance between them. However, the position of the nodes on the inclined layers is determined by a series of transformations and in some cases by a minimization function. Thus, nodes from one layer may penetrate another geometrically neighboring surface. Moreover, within the nodes in the inclined layer itself, the position of all nodes is determined by the position of the bounding nodes (which are the four nodes bounding the inclined layer. Therefore, while checking for penetration, the bounding nodes, as well as the nodes lying on the boundary should be the primary contenders. Since we expect the penetration to be small in practice, the working algorithm limits the search to the nodes lying on the boundary. The algorithm works well on the meshes we have tried, though it may not be complete. It works as follows:

- Find all contender nodes, i.e. nodes which lie on the boundary of two layers
- For each of these nodes
  - Find ALL joined elements on both the layers containing a candidate node.
  - Find neighboring layers and their planes
  - For each neighboring plane
  - Check whether there is AT LEAST one joined element from each of the containing layers, which is split by the neighboring plane. If there are such elements, declare Penetration.
  - Do the following only if there is penetration
  - Determine the side of the neighboring plane the penetrating node should lie on and perturb it to that side in a direction perpendicular to the plane, such that it clears the plane by twice the contact thickness.

5.6 Convexity Check

Our algorithm assumes that the quadrilateral elements used in the mesh are convex, and a violation of this assumption may cause the algorithm to produce inconsistant results. Thus, a module for checking the convexity of the elements has been implemented to check the convexity of the mesh at any stage. Usually, this check is done at the end of each fold, and if an element is found to be concave, it is split into two triangles, thereby curing the mesh of unwanted concave elements. The “convexity check” routine essentially implements the well-known algorithm, which checks a planar polygon for convexity by checking the direction of each turn of the vertex of the polygon. An added issue is that the quadrilaterals in the mesh

produced by the folding algorithm may not always be planar and be slightly warped. As the warping is usually small, the convexity algorithm is used by projecting the quadrilateral onto a best fit plane.

6 CASE STUDY

In this section the working of the folder is illustrated by the means of a case study. This example illustrates the functioning as well as the efficacy of the algorithm. The generated mesh is compared with the results from one other commercially available software for airbag folding, SAFE EDITOR^{TM}. The example on which we perform the study is a typical passenger side airbag. The geometric details have been measured from a commercially available airbag. The initial state of the airbag is a closed disc of 640 mm diameter, with two planes of the disc separated by a distance of 0.4 mm, and connected by elements throughout the circumference. This is the initial state of the fabric of the airbag before the start of the folding process and is shown in Figure 16. Here the geometry of the two layers of the airbag is identical, which is not a necessity for the folder described here, but is essential for SAFE EDITOR^{TM}. In fact, SAFE EDITOR^{TM} asks for a planar geometry as input to the folder and assumes that the second layer is identical to this.

Images from a particular folding sequence are shown in Figure 17 to Figure 29, where a single layer (circular mesh) was folded four times.

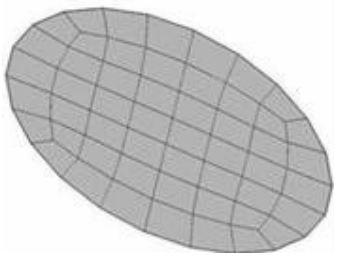

Figure 16 The initial airbag geometry in the form of a disc

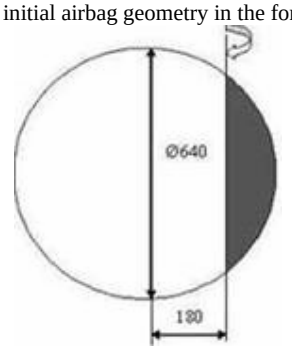

Figure 17 The schematic of the first fold

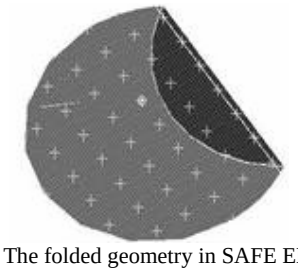

Figure 18 The folded geometry in SAFE EDITOR^{TM}

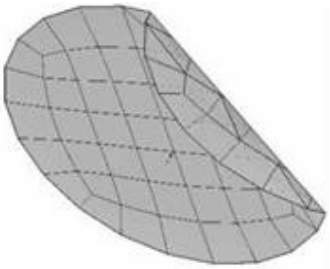

Figure 19 The folded mesh generated by our algorithm

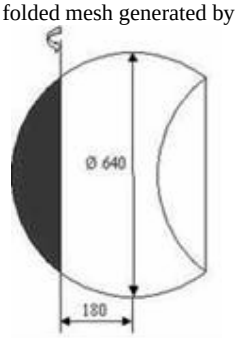

Figure 20 The schematic of the second fold

Figure 21 The folded geometry in SAFE EDITOR^{TM}

Figure 22 The folded mesh generated by our algorithm

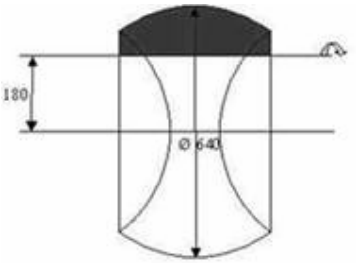

Figure 23 The schematic of the third fold

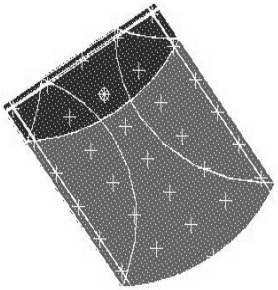

Figure 24 The folded geometry in SAFE EDITOR™

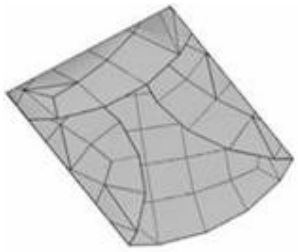

Figure 25 The folded mesh generated by our algorithm.

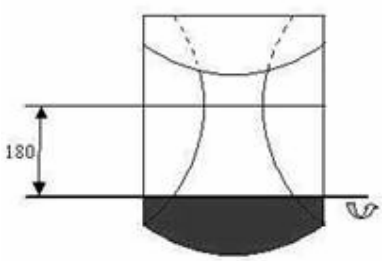

Figure 26 The schematic of the fourth fold

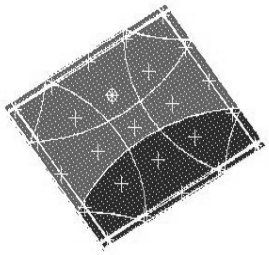

Figure 27 The folded geometry in SAFE EDITOR™

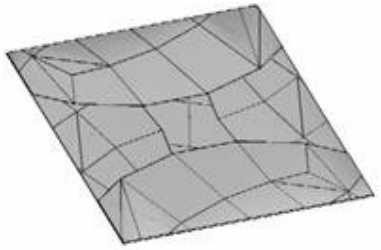

Figure 28 The folded mesh generated by our algorithm.

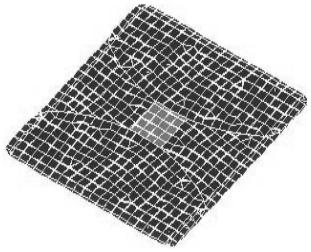

Figure 29 The generated mesh on the airbag folded with SAFE EDITOR™

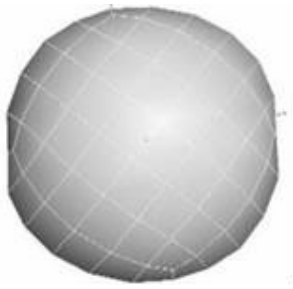

Figure 30 The original shape of the airbag

7 RESULTS

The folding process is significant only during the deployment sequence. Ideally the flat unfolded bag once inflated should have the same shape, surface area and volume as the inflated airbag. The parameters used to measure the performance of the algorithm are hence change in surface area of the bag during folding, change in volume of the inflated bag before and after folding, observed time steps in the inflation simulation and the visual shape of the bag. The algorithm proves to be very efficient in controlling the change in area of the airbag while folding as is demonstrated by fact that the change in surface area with respect to the initial airbag after these four folds is only 0.11% (0.637995 m^2 from the initial 0.63874 m^2). Whereas, the surface area in the airbag folded using SAFE EDITOR™ changes by almost 11% during these four folds to 0.570521 m^2. The unfolded airbag when inflated occupies a volume of 48.8 litres, which is reduced to 48.1 litres after four folds using the proposed software. The same airbag outputs a volume of 42.7 litres if folded using SAFE EDITOR™. These results are summarized in **Table 1**. The reasons for the improved performance of the algorithm can be inferred from a visual inspection of the shape of the inflated bags (shown in Figure 31 to Figure 34) after being folded by the two packages. The bag folded using SAFE EDITOR™ undergoes a visible change in shape, while the bag folded by the proposed algorithm retains its similarity to the original shape, shown in Figure 30.

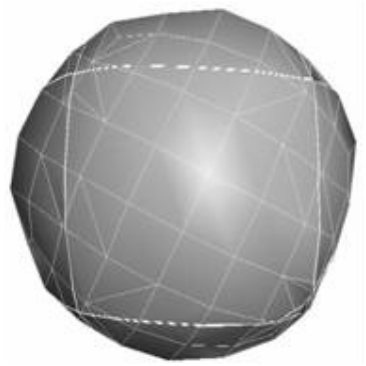

Figure 31 Figure Top view of the inflated airbag, which was folded with the presented software

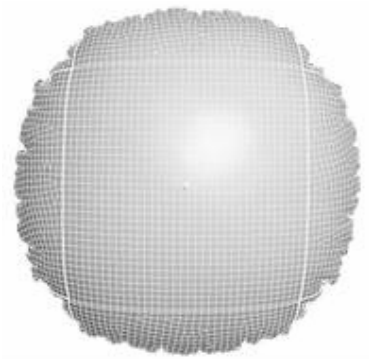

Figure 32 Figure Top view of the inflated airbag, which was folded with SAFE EDITOR™

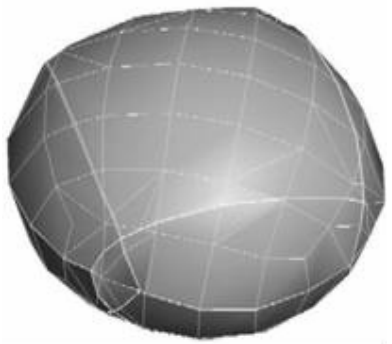

Figure 33 Isometric view of the inflated airbag, which was folded with the presented software

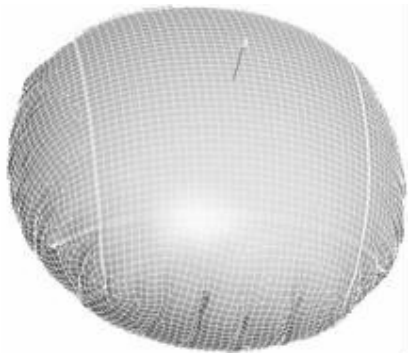

Figure 34 Isometric view of the inflated airbag

The inflation process of both the airbags was carried out on PAM-CRASH™ software with the same simulation parameters. On comparing the time steps in the simulation, it is observed that there are more than 7000 elements in the airbag folded with the SAFE EDITOR™ and the lowest time step is of the order of 1e-6. All the elements have a time step of the order of 1e-5 or lower. In the airbag modeled using the proposed software, there are only about 550 elements, with about 300 having a time step lower than 1e-5, resulting in a much faster simulation than the SAFE EDITOR™ airbag. This difference is due to

the fact that this algorithm refines the airbag mesh only where it is required and thus the mesh remains overall well behaved. As is visible in Figure 31, the mesh output from the software remains coarse where it is not required for it to be refined, while the airbag modeled with SAFE EDITOR^{TM} is too fine even in places that do not require refinement. Hence, the run time for inflation using mesh generated using the proposed algorithm is lower and the inflation process is also more stable. as the enhanced stability is in the sense that as the number of folds of the airbag increase, our algorithm is found to generate fewer elements with small surface area and a bad aspect ratio as compared to the SAFE EDITOR^{TM} algorithm..

Algorithm used	Change in surface area	Change in Volume
SAFE EDITOR^{TM}	11%	12.5%
Our algorithm	0.1%	1.5%

Table 1 Comparison of algorithm performance

8 LIMITATIONS

There are certain configurations of the folding plane passing through the transient patch which cannot be accommodated by the algorithm. As these positions are quite limited and usually a small translation of the folding plane would make it admissible to the software, this does not affect the functionality of the algorithm as far as airbag modeling is concerned.

9 CONCLUSIONS

In this paper, a new approach to airbag modeling for FE simulations is presented, which preserves the geometry and surface area of the airbag during the folding process. A case study on a typical airbag is conducted using the presented software, and it is found that the proposed approach compares favorably with existing software and preserves the geometry of the airbag during the folding process. This software finds applications in airbag modeling for FE simulations, and is optimized for the same.

10 REFRENCES

[1] Bridson, R., Fedkiw, R. and Anderson, J., "Robust Treatment of Collisions, Contact and Friction for Cloth Animation", ACM Transactions On Graphics, 2002, 21, 594–603.

[2] Bridson, R., Marino, S., and Fedkiw, R., "Simulation of Clothing with Folds and Wrinkles", Proceedings of the ACM SIGGRAPH / Eurographics Symposium on Computer Animation, San Diego, California, Eurographics Association, 2003.

[3] Chawla A, Mukherjee S, and Sharma A, Development of FE meshes for folded airbags, International Journal of Crashworthiness, 2005, Vol 10, No 3, pp 259-266.

[4] Chawla A, Mukherjee S and Sharma A, Mesh Generation for folded airbags, Computer Aided Design and Applications, Vol 1 No 1-4, P269-276, 2004.

[5] Chawla A, Mukherjee S and Sharma A, Mesh Generation of folded airbags, Proceedings of the International CAD conference, May, 2004, Pattaya, Thailand.

[6] Dongliang Zhang, and Matthew M. F. Yuen, “Cloth simulation using multilevel meshes”, Computers and Graphics, 2001, 25, 3, 383–389.

[7] PAM-Generis^{TM} manual, ESI Group Software Product Co. Paris, 2000.

[8] Safe-tutorial, The Safe editor manual, ESI Group Software Product Co. Paris, 2000.

[9] Liu, J., Ko, M., and Chang, R., “A simple self collision avoidance for cloth Animation”, Computers and Graphics, 1998, 22, 1, 117–128.

[10] Ng, N., and Grimsdale, R.L., “Computer graphics techniques for modeling cloth”, Computers and Graphics, 1995, 19, 3, 423–430.

[11] Ng, N., Grimsdale, R.L., Allen, W., “A system for modeling and visualization of cloth material”, Computers and Graphics, 2001, 25, 3, 383–389.

List of Captions